Holistic Prediction on a Time-Evolving Attributed Graph

Shohei Yamasaki
Osaka University

Yuya Sasaki
Osaka University

Panagiotis Karras
Aarhus University

Makoto Onizuka
Osaka University

ABSTRACT

Given a time-evolving attributed graph, can we predict its future links, nodes, and attributes? Existing techniques assume that each prediction task is independent of others, and do not try to predict the appearance of new nodes that were not observed in the past.

In this paper, we address two interrelated questions; (1) can we exploit the interdependence between tasks to improve prediction accuracy? and (2) can we predict new nodes with their attributes? We answer these questions by proposing a unified framework that predicts both node attributes and topology changes such as the appearance and disappearance of links and the emergence and loss of nodes. This framework comprises components for independent and interactive prediction, including methods for predicting new nodes. Our experimental study using real-world data confirms that our interdependent prediction framework achieves higher accuracy than methods based on independent prediction.

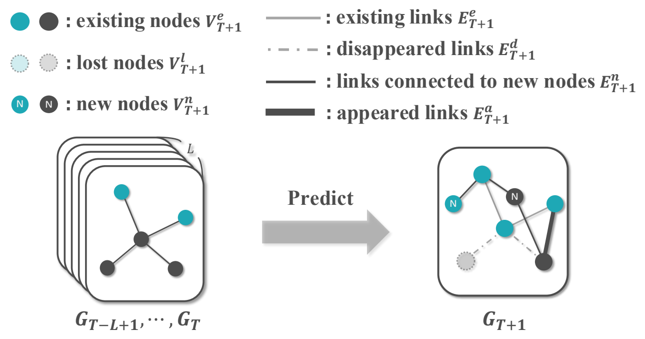

: existing nodes V^e_{T+1}
: lost nodes V^l_{T+1}
: new nodes V^n_{T+1}
: existing links E^e_{T+1}
: disappeared links E^d_{T+1}
: links connected to new nodes E^n_{T+1}
: appeared links E^a_{T+1}
L
Predict
N
$G_{T-L+1}, \cdots, G_T$
G_{T+1}

Figure 1: An example of time-evolving attributed graph prediction; node colors represent attributes. T and L indicate the current time step and the number of time steps, respectively.

1 INTRODUCTION

Real-world connections and interactions, such as those in social networks [14] and co-author networks [30] are often modeled by graphs where nodes represent objects and edges represent relationships. The structures and node attributes of such graphs often evolve over time by operations such as node addition/deletion, link addition/deletion, and node attribute changes. For instance, social networks change over time in terms of participating users, their profiles, and their friendship links. Such temporally malleable graphs are called *time-evolving attributed graphs* [23, 27].

Various applications [15, 39] call for *predicting* the future evolution of a time-evolving attributed graph. This prediction task is composite, including several sub-prediction tasks, such as attribute prediction and link prediction [11, 19]. Conventionally, this composite prediction task is addressed in a *compartmentalized* manner, separating it into independent sub-prediction tasks and separately applying a technique for each component task. Unfortunately, this compartmentalized approach treats component prediction tasks separately rather than as an interdependent whole and utilizing one prediction to inform another. Besides, to our knowledge, previous works in time-evolving graph prediction do not predict the appearance of new nodes and their attributes. However, in many time-evolving attributes graphs, both task interdependence and new node appearances are prevalent.

Example 1.1. Let us consider the evolution of a social network. Newly registered accounts acquire connections to existing accounts with whom they have similar profiles. When accounts are deleted, accounts connected to the deleted accounts may connect to other accounts similar to the deleted ones. Thus, predictions of deletion, registration, and connection of accounts are interdependent.

In this paper, we introduce the problem of holistically predicting the future in a time-evolving attributed graph. Figure 1 illustrates the problem, which includes multiple sub-prediction tasks, such as predicting node loss and new node appearance. To properly address this problem and achieve high prediction accuracy, we need to capture the interdependence between sub-prediction tasks. However, as it is difficult for a single method to capture all interdependences, we must effectively combine several methods, where we interactively reuse the results of one task to predict other tasks. Furthermore, as existing methods cannot effectively predict new node appearances, we need novel prediction methods for this sub-problem.

We propose AGATE (*A General framework for predicting Attributed Time-Evolving graphs*), a holistic, versatile prediction framework that leverages the interdependence between the tasks of predicting new nodes, lost nodes, appearing links, disappearing links, and node attributes from observed past graphs. AGATE comprises components for predicting the future graph and capturing task interdependence: (i) one to predict the size of an evolving graph, (ii) one to predict existing-node structure and attributes, as well as new nodes, and (iii) a *reuse mechanism* that reuses the results of (ii) to achieve predictions that capture the interdependence between sub-predictions. Our framework is modular; it can use existing methods as components for prediction tasks. We develop two novel prediction methods, TGGNN for predicting existing node and links and PROSER for predicting new node appearance with attributes.

We evaluate AGATE vs. state-of-the-art methods using three real-world time-evolving attributed graphs. Our experiments show that AGATE benefits by allowing prediction tasks to affect each other, while its reuse mechanism improves prediction accuracy, and it accurately predicts new nodes with attributes.

2 PRELIMINARY

An undirected *attributed graph* is a triple $G = (V, E, X)$ where V is a finite set of *nodes*, $E \subseteq V \times V$ is a set of *edges*, and $X \in \mathbb{R}^{|V| \times d}$ is a set of node attributes; $X^{(v)}$ denotes d dimensional attributes of node v. Each value in X may take both categorical and numerical values. We consider an *evolving* graph, in which the numbers of nodes and links and the values of node attributes dynamically change over time steps, while the number of attributes d remains constant. We define time-evolving attributed graphs as follows:

Definition 1. *A* time-evolving attributed graph *is a sequence of attributed graphs $\langle G_1, \cdots, G_T \rangle$ over discrete time steps, where T is the number of observed time steps and $G_t := (V_t, E_t, X_t)$ the attributed graph at time step t.*

In a time-evolving graph, nodes and links appear and disappear dynamically. We distinguish three types of nodes at time step t: *existing* nodes V_t^e, *new* nodes V_t^n, and *lost* nodes V_t^l, as follows:

$$V_t^e = V_t \cap V_{t-1}, \qquad V_t^n = V_t \setminus V_{t-1}, \qquad V_t^l = V_{t-1} \setminus V_t.$$

Intuitively, nodes in V_t^e exist at both time steps t and $t-1$, those in V_t^n appear at time step t, and those in V_t^l disappear at time step t. Likewise, we distinguish among existing links E_t^e, links connected to new nodes E_t^n, appearing links E_t^a, and disappearing links E_t^d:

$$E_t^e = E_t \cap E_{t-1}, \qquad E_t^n = E_t \setminus (V_{t-1} \times V_{t-1})$$
$$E_t^a = (E_t \setminus E_{t-1}) \setminus E_t^n, \qquad E_t^d = (E_{t-1} \setminus E_t)$$

3 PROBLEM STATEMENT

We define the problem as follows:

Problem 1. *Given a time-evolving attributed graph, $\langle G_{T-L+1}, G_{T-L+2}, \ldots, G_T \rangle$, including appearing and disappearing nodes and links and changing node attributes, the* **holistic time-evolving attributed graph prediction** *problem asks to predict the graph at time step $T+1$, $G_{T+1} = (V_{T+1}, E_{T+1}, X_{T+1})$.*

This problem calls to predict the whole of G_{T+1}, rather than one of its components only, and involves sub-prediction tasks.

3.1 Sub-prediction tasks

Here, we define each sub-prediction task. Note that existing works address some of these sub-prediction tasks independently.

3.1.1 Prediction on existing nodes. These tasks aim to predict graph structure and attributes on existing nodes. Such tasks have been addressed in prior studies, such as STGCN [39] and DynGEM [8]:

We have six sub-prediction tasks; node loss, link appearance, link disappearance, attribute changes on existing nodes, and attribute values on existing nodes.

SubProblem 1. (**Node loss**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, this sub-prediction task is to predict the nodes V_{T+1}^l that are lost from V_T.*

SubProblem 2. (**Link appearance**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, this sub-prediction task is to predict links appearing between any pair of nodes $u, v \in V_T$, which also exist at time $T+1$ and are not connected at time step T.*

SubProblem 3. (**Link disappearance**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, this sub-prediction task is to predict links disappearing between any pair of nodes $u, v \in V_T$, which exist at time step $T+1$ and are connected at time step T.*

SubProblem 4. (**Attribute *changes* on existing nodes**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, this sub-prediction task is to predict whether a categorical attribute of an existing node changes from T to $T+1$.*

SubProblem 5. (**Attribute values on existing nodes**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, this sub-prediction task is to predict a new attribute value on an existing node at time $T+1$.*

3.1.2 Prediction on new nodes. This group of tasks aims to predict new nodes with their links and attributes. Link prediction on new nodes has been addressed recently in [11], but there exist no studies for the prediction of new node attributes.

SubProblem 6. (**New node attributes**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, this sub-prediction task is to predict new nodes $\hat{V}_{T+1}^n$ and their attributes $\hat{X}_{T+1}^n$.*

As there is no previous work on the prediction of new nodes with attributes, we define an evaluation measure for this task. We evaluate the similarity between the sets of attributes of predicted and real new nodes based on a perfect bipartite matching [31]:

Definition 2. (**Perfect matching**): *Given a complete bipartite graph $K = (V, V', E)$ where $E = V \times V'$, a* perfect matching *M in K is a set of pairwise non-adjacent edges that cover all vertices in V.*

We construct a perfect matching K from predicted to real new nodes, and compute edge weights by cosine similarity:

$$K = (\hat{V}_{T+1}^n, V_{T+1}^n, \hat{V}_{T+1}^n \times V_{T+1}^n), \quad w((\hat{v}, v)) = \frac{\hat{X}_{T+1}^{(\hat{v})} \cdot X_{T+1}^{(v)}}{\|\hat{X}_{T+1}^{(\hat{v})}\| \|X_{T+1}^{(v)}\|}$$

where $X_{t+1}^{(v)}$ denotes the attribute of node v at time step $t+1$.

Our measure is the maximum similarity:

$$\text{M-sim}(\hat{V}_{T+1}^n, V_{T+1}^n, \hat{X}_{T+1}^n, X_{T+1}^n) = \max_{M \in \mathcal{M}} \frac{1}{|M|} \sum_{(\hat{v}, v) \in M} w((\hat{v}, v)) \quad (1)$$

where $\mathcal{M}$ is the set of perfect matchings for the given bipartite graph. If the similarity is closer to one, attributes on predicted new nodes are similar to those on corresponding real new nodes. By this measure, we enforce that predicted attribute values properly match real ones, rather than merely reiterate values that appear frequently in new nodes of G_{T+1}.

When it comes to predicting links to new nodes, prior work [11] assumes that new nodes are *given*, along with their attributes. Contrariwise, we predict links to *predicted* new nodes.

SubProblem 7. (**Links connected to new nodes**): *Given a time-evolving attributed graph $\langle G_{T-L+1}, \ldots, G_T \rangle$ across L time steps, new nodes $\hat{V}_{T+1}^n$, and corresponding attribute values $\hat{X}_{T+1}^n$ predicted by subproblem 6, this sub-prediction task is to predict any link between v and $\hat{v}$, $(v, \hat{v}) \in V_T \times \hat{V}_{T+1}^n$.*

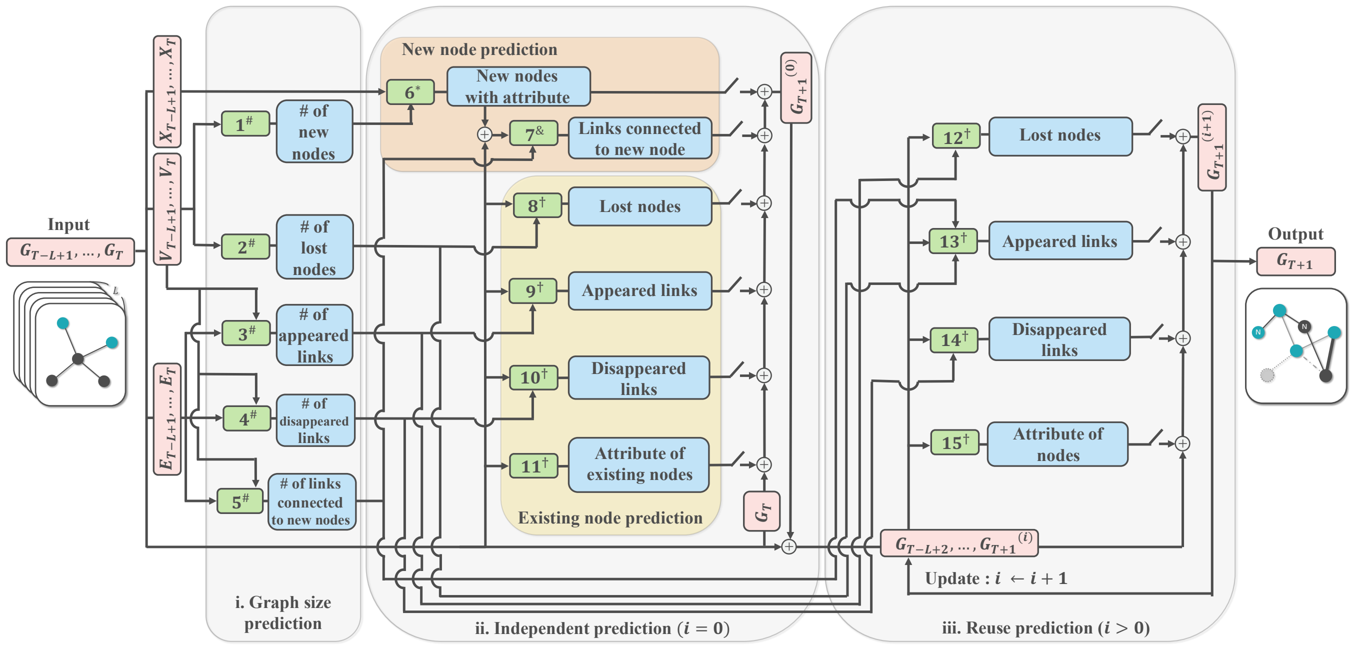


Figure 2: AGATE architecture; input: observed graphs (time $L-T+1$ to T); output: predicted graph (time $T+1$); pink: graph data; green: learning models; blue: predictions. Annotations (#, $*$, &, †) to models (1–15) point to candidate methods as follows: # can be any time-series forecasting methods such as LSTM [12]; $*$ can be any method for predicting attributes of new nodes such as PointNet [25] and PROSER; & can be any method for predicting links connected to new nodes such as FNN and DEAL [11]; † can be any method for predicting graph structures and attributes of existing nodes such as EvolveGCN [23] and TGGNN.

3.2 Task interdependence

The sub-prediction tasks may be interdependent; for example, attribute prediction affects link prediction and vise versa. The way tasks affect each other is unknown, and the interdependence between predictions regarding new nodes and those regarding existing nodes is uncertain. Such interdependence may depend on the particular data. Nevertheless, we venture to explore such task interdependence and exploit it to any extent deemed possible.

4 AGATE: OUR FRAMEWORK

AGATE holistically predicts the future of time-evolving attributed graphs, exploiting the interdependence of sub-prediction tasks. Figure 2 illustrates its architecture. AGATE consists of three main components for graph size prediction, independent prediction, and reuse prediction; the independent prediction component is further subdivided in parts for existing and new nodes.

AGATE first predicts the output graph size characteristics. Thereafter, it first conducts each sub-prediction task independently to obtain a preliminary predicted graph at time step $T+1$, and then updates the results of each such task by reusing the results of other tasks. AGATE is modular; it can employ any existing method as a component in both independent and reuse predictions; however, to our knowledge, as no existing method supports the prediction of new nodes with attributes, we develop our own methods for this task. Furthermore, AGATE is general, as it can support prediction in attributed graphs that are either partially or fully time-evolving; for example, it can accommodate static attributes or no link disappearance. In the following, we explain each component.

4.1 Graph size prediction

In case the methods used in individual sub-prediction tasks determine the numbers of nodes and links in the output graph as well, we do not need a separate prediction on graph size. Otherwise, we utilize a separate component to predict the sizes of V_{T+1} and E_{T+1}; we extract sequences of new nodes $\{V^n_{T-L+1}, \dots, V^n_T\}$, lost nodes $\{V^l_{T-L+1}, \dots, V^l_T\}$, appearing links $\{E^a_{T-L+1}, \dots, E^a_T\}$, disappearing links $\{E^d_{T-L+1}, \dots, E^d_T\}$, and links to new nodes $\{E^n_{T-L+1}, \dots, E^n_T\}$ from input graphs $\{G_{T-L+1}, \dots, G_T\}$, count the elements (e.g., lost nodes) in each sequence, normalize them through division by its maximum value to avoid vanishing gradient, and train the model using the sequence of numbers of elements by time-series data prediction, such as LSTM [12].

4.2 Independent prediction

This component makes initial predictions, which may then be used in reuse prediction; independent prediction is subdivided into parts regarding existing nodes and new nodes.

4.2.1 Existing node prediction. This component involves binary classification tasks regarding node loss, link appearance, link disappearance, as well as multi-class classification and regression tasks regarding the attributes of existing nodes. We obtain a probability for each target node/link from each classification model and derive the predicted results as a set of highest probability elements with cardinality given by the respective size prediction module. In node attribute prediction, we can select models to either predict each attribute value individually, or all values simultaneously. We use suitable models (e.g., LSTM [12] and DynGEM [8]) to predict categorical and numerical node attributes. To our knowledge, only EvolveGCN [23] supports all tasks on existing nodes. However, EvolveGCN performs poorly compared to other models (see Table 2). In Section 5.1 we present our own novel method, TGGNN.

4.2.2 New node prediction. As the literature lacks methods to predict the attributes and links of new nodes, we develop three baseline methods for this purpose (Random, FNN, and PointNet), which aim to determine the correspondence between the predicted and real new nodes by maximizing the matching similarity in Equation 1: **Random** outputs randomly sampled nodes from V_T^n with attributes from X_T^n. **FNN** is a simple feedforward neural network, i.e., a multilayer perceptron, trained by constructing a bipartite graph from predicted new nodes $\hat{V}_{T+1}^n$ to real new nodes V_{T+1}^n, and learning to minimize the loss function: $\mathcal{L}_{n,v} = -\log \frac{\text{M-sim}(\hat{V}_{T+1}^n, V_{T+1}^n, \hat{X}_{T+1}^n, X_{T+1}^n)+1}{2}$. FNN's learning depends on the order of input nodes. **PointNet** is a deep learning framework that deals with order invariance [25], with input and loss function being the same as those of FNN.

We utilize the predicted new node attributes to predict links to new nodes using DEAL [11], which can handle link prediction for nodes having only attribute information, or any other similar method. We evaluate prediction accuracy by the correspondence between predicted and real nodes.

4.3 Reuse prediction

This component updates prediction results by exploiting task interdependence, aiming to capture the effect of each sub-prediction on others. The reuse prediction follows the specifications of existing node prediction, yet now each model reuses the already predicted graph characteristics, so as to update the results of all sub-prediction tasks. We may use some of the independent prediction results, as we see fit; we study this matter in our experiments.

5 NOVEL PREDICTION METHODS

Our preliminary experiments revealed that, unfortunately, existing methods fare poorly in future graph prediction. To obtain better prediction accuracy, we develop two novel methods: the *Temporal Gated Graph Neural Network* (TGGNN) for predictions involving existing nodes and the *Probabilistic Selection Rule* (PROSER) for the prediction of new node attributes.

5.1 TGGNN

We design TGGNN as a novel node representation learning model taking into account the nature of sub-prediction tasks: first, sub-prediction tasks are inductive, since test data graphs may be different from those in the training data; second, the topologies of graphs $G_{T-L+1}, \ldots, G_T$ may differ from each other, hence we need to handle topological changes as well as attribute changes; therefore, TGGNN supports inductive tasks on time-evolving attribute graphs whose nodes, links, and node attributes change dynamically.

While graph neural networks perform well for node representation learning, they assume fixed topologies, hence do not support time-evolving graphs with changing topologies. We extend graph neural networks to support induction on graphs with time-dependent topologies and attributes. TGGNN learns node embeddings in G_T that capture both topological and attribute changes in the time-evolving attributed graph, informed by the local *dynamic* graph structure, rather than the global static graph structure; in particular, it aggregates embeddings within each node's k-hop neighborhood in all observed time steps ($G_{T-L+1}, \ldots, G_T$) and captures node attributes by learning temporal attribute changes as graph annotations [18] independently of graph structure.

TGGNN utilizes a suite of new gating mechanisms, ST-Gate, which improves the ST-Convolution Block [39] so as to handle topological changes. ST-Gate first applies time-directed convolution [5] on L graphs and then aggregates hidden states of nodes and their neighbors at each time step by GRU-like updates [4]. In parallel, it applies time-directed convolution on graph annotations [18], which are initially concatenations of node attributes over L time steps; then it aggregates the embeddings of the aggregated hidden states and the graph annotation and applies one more time-directed convolution [5] on the aggregated embedding.

The inputs to TGGNN are $\mathcal{A} \in \mathbb{R}^{n \times L \times d}, \mathcal{H} \in \mathbb{R}^{n \times L \times \hat{d}}$ and $\mathcal{E} \in \mathbb{R}^{n \times L \times n}$ where n is the total number of nodes in the observed graphs and $\hat{d}$ the hidden state dimensions; $\mathcal{A}$ denotes a time-series of node attributes, $[X_{T-L+1}; \ldots; X_T]$ as graph annotations, where $[;]$ means concatenation; $\mathcal{H}$ denotes the hidden states and $\mathcal{E}$ a time-series adjacency matrix derived from $\{E_{T-L+1}, \ldots, E_T\}$. We initialize hidden state $\mathcal{H}^{(0)}$ using the graph annotations $\mathcal{A}^{(0)}$; we may pad with extra 0s to allow hidden states that are larger than the annotation size.

Let $\mathcal{H}^{(N)} \in \mathbb{R}^{n \times (L-2N(K-1)) \times \hat{d}}$ be the hidden state output of the Nth ST-Gate, where K denotes the kernel size of time-directed convolution operators in the ST-Gate. The final node embedding output $\mathcal{H}^{(\text{out})}$ of TGGNN is calculated as:

$$\mathcal{H}^{(\text{out})} = \phi(\{(\mathcal{H}^{(N)} * \Gamma_f + b_f) \otimes \sigma(\mathcal{H}^{(N)} * \Gamma_g + b_g)\}W + b) \in \mathbb{R}^{n \times D}$$

where $\Gamma_f, \Gamma_g \in \mathbb{R}^{(L-2N(K-1)) \times \hat{d} \times \hat{d}}$ are 1-dimensional convolution operators for the final embedding whose kernel size is $L-2N(K-1)$ and $b_f, b_g \in \mathbb{R}^{\hat{d}}$ are correspondence biases; $\otimes$ denotes the element-wise multiplication. $W \in \mathbb{R}^{\hat{d} \times D}, b \in \mathbb{R}^D$ are learnable weights and bias for a linear transformation and ϕ denotes any of the activation function suited for the prediction task; D is the size of predicts (e.g., $D = n$ on link prediction). Since the TGGNN output is computed from temporal graph structures and node attributes, it captures the evolution of a dynamic attributed graph.

5.2 PROSER

PROSER aims to maximize the matching similarity in the prediction of new node attributes. Through a preliminary analysis, we observed that most new nodes have similar attributes to those of new nodes at the previous time step. Thus, prediction accuracy is relatively high when we randomly sample attributes of new nodes at the previous time step. We refine this random sampling by selecting appropriate nodes, so as to increase the matching similarity.

Table 1: Data statistics; we indicate the numbers of unique nodes and edges across all time steps.

	# Nodes	# Edges	# Attributes	# Time Steps
NBA	3,781	95,203	35	66
Reddit	7,756	23,554	300	14
AMiner	109,87	6,708	9	24

The rationale of PROSER is to improve upon simple random sampling, whereby matching similarity is low when the cosine similarity of sampled attributes to any attributes of new nodes at time step $T+1$ is low. PROSER avoids selecting such attributes, by estimating the highest cosine similarity between sampled attributes and those of new nodes at time step $T+1$. Besides, matching similarity would not increase by frequently sampling a few attributes that are highly similar to some real node attributes, as, in a perfect matching, each match is unique. We need to sample attributes X^n_T having a similar distribution to that of new node attributes in time $T+1$, X^n_{T+1}. We design PROSER with this objective in mind.

PROSER employs logistic regression to learn probabilities that cosine similarities between sampled and actual new node attributes are higher than a threshold θ. Before model training, we determine θ to maximize matching similarity, as:

$$\theta = \arg\max_{\theta_i \in \mathbb{R}} \sum_{G_i \in \mathcal{T}} \text{M-sim}(V^n_i(\theta_i), V^n_{i+1}, X^n_i(\theta_i), X^n_{i+1}) \tag{2}$$

where $\mathcal{T}$ denotes the set of training graphs, $V^n_i(\theta_i)$ is the set of sampled attributes whose cosine similarity to those of the real nodes is higher than θ_i and $X^n_i(\theta_i)$ is the set of their attributes; we use θ to remove nodes that do not increase the matching similarity. To capture the distribution, we use a mean vector of X^n_T as a representative attribute of G_T. Our model computes the probability to maximize the matching similarity for future graphs by leveraging this mean and the threshold, without using the graph at time $T+1$.

In the inference phase, PROSER randomly samples attributes from the set of new node attributes at time T, and calculates probabilities. If these probabilities are higher than a user-specified threshold, we add these attributes to prediction results; otherwise, we discard them. We repeat, until the number of predicted attributes reaches the predicted number of new nodes. Since we randomly select candidate predicted attributes, our predictions are diverse.

6 EXPERIMENTS

We conduct a thorough experimental study to validate: (1) the individual performance of TGGNN and PROSER, (2) the performance gain due to the exploitation of task interdependence, (3) the accuracy of new node predictions. Our source codes and data are available online [6]. All experiments are conducted on a Linux server with Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz.

6.1 Datasets

We use three datasets with diverse time-evolving features: NBA [21], Reddit [26], and AMiner [1]. Table 1 shows statistics on the data.

NBA: This dataset includes statistics about NBA players. Nodes represent players, with node attributes consisting of points, team, age, and position. Two nodes are linked when the respective players are teammates. Nodes, links, and attributes change dynamically.

Reddit: This dataset is a subreddit hyperlink network; nodes stand for subreddits and links represent posts between them; if a subreddit has no posts in a time step, the corresponding node is removed at that time step. Node attributes are embeddings of the subreddits [14]. Nodes and links change dynamically, while node attributes are static across time.

AMiner: This is a citation network extracted from AMiner [30]. Nodes stand for papers, links for citations, and attributes for research field weights, extracted by computing TF-IDF scores from paper titles. Nodes and links appear over time, but nodes do not disappear, while the links and attributes of existing nodes are static.

We divide time steps into training, validation, and test data with a 7:2:1 ratio; in NBA data, that is time steps 1–48 for training, 49–60 for validation, and 61–66 for test; in Reddit, that is time steps 1–11 for training, 12–13 for validation, and 14 for test; in AMiner, time steps 1–20 for training, 21–22 for validation, and 23–24 for test.

6.2 Compared methods and evaluation metrics

We apply different methods for comparison in each sub-prediction task, since not all methods are applicable to all tasks. When predicting existing node features, we use 9 methods grouped in four categories: simple, dynamic graph embeddings, static graph GNNs, and dynamic graph GNNs. Simple methods are: (1) **Baseline**, which outputs G_T as the predicted graph for G_{T+1}, (2) **Random**, which randomly samples the predicted targets, (3) **FNN**, and (4) **LSTM** [12]. We use (5) **DynGEM** [8] as a dynamic graph embedding and (6) **GCN** [13] as a static graph GNN. As dynamic graph GNNs, we use (7) **STGCN** [39], (8) **EvolveGCN** [23], which has two versions, EvolveGCN-H and EveloveGCN-O, and (9) our own method, **TG-GNN**; dynamic graph GNNs are the state-of-the-art methods.

To predict new nodes with attributes, we use the three baseline methods of Section 4.2.2 and PROSER, presented in Section 5.2. To predict links connected to new nodes, we use cosine similarity (CS for short), FNN, and DEAL [11] based upon the results predicting new nodes with attributes. DEAL is the state-of-the-art method.

AGATE repeatedly predicts G_{T+1}; we identify the best model for each sub-prediction task on the validation data, and then predict the graphs by reusing the best model results.

We use standard metrics for all sub-prediction tasks. In tasks involving existing nodes and links to new nodes, we use the area under the ROC curve (AUC) and average precision when it comes to binary classification (categorical values), and mean absolute error (MAE) and root mean squared error (RMSE) when it comes to regression (numerical values). In tasks involving new node attributes, we use the mean, median, max, and min cosine similarities of the perfect match between the predicted and real new nodes.

6.3 Experimental results

We present results for independent prediction methods and for AGATE, which exploits task interdependence. We start with results on predictions involving existing nodes, and continue with results on predictions about new nodes.

Prediction on graph size: Table 4 shows mean absolute error of baseline and LSTM on the graph size prediction. LSTM generally performs better than baseline; whereas, baseline performs well when the size of graphs is relatively stable.

Table 2: Results on node/link prediction; underlined fonts indicate the best results in independent prediction; bold fonts indicate the best results among both independent and reuse predictions.

Methods	Node prediction (loss)				Link prediction (appearance)				Link prediction (disappearance)			
	ROC AUC		Average precision		ROC AUC		Average precision		ROC AUC		Average precision	
	NBA	Reddit	NBA	Reddit	NBA	Reddit	NBA	Reddit	NBA	Reddit	NBA	Reddit
Baseline	0.5000	0.5000	0.1871	0.5110	0.5000	0.5000	<u>0.0137</u>	0.0005	0.5000	0.5000	0.4136	0.3929
Random	0.5141	0.5138	0.1916	0.5181	0.4988	0.5116	0.0136	0.0005	0.5020	0.5027	0.4146	0.3942
FNN	0.8533	0.7750	0.5358	0.7456	0.4988	0.4838	0.0135	0.0005	0.4949	0.4876	0.4104	0.3777
LSTM	0.8547	0.8264	0.5394	0.8163	0.5003	<u>0.8271</u>	0.0135	**<u>0.0572</u>**	0.4887	<u>0.7194</u>	0.3999	<u>0.5664</u>
DynGEM	0.4731	0.6735	0.1824	0.6344	0.4945	0.4551	0.0136	0.0004	<u>0.5106</u>	0.4625	<u>0.4253</u>	0.3518
GCN	0.5743	0.7506	0.2453	0.7196	<u>0.5014</u>	0.8253	0.0136	0.0202	0.5062	0.6308	0.4093	0.4751
STGCN	0.6007	0.8222	0.2686	0.8072	0.4937	0.7359	0.0132	0.0011	0.4883	0.6530	0.4013	0.5039
EvolveGCN-H	0.5742	0.6131	0.2211	0.5791	0.4932	0.5652	0.0134	0.0007	0.4865	0.5109	0.4005	0.3982
EvolveGCN-O	0.5719	0.6460	0.2313	0.6052	0.4928	0.5110	0.0135	0.0005	0.4762	0.5252	0.3936	0.4116
TGGNN	<u>0.8619</u>	<u>0.8284</u>	<u>0.5667</u>	<u>0.8195</u>	0.5007	0.7664	0.0135	0.0014	0.5034	0.6218	0.4072	0.4741
AGATE	**0.9520**	**0.8930**	**0.7889**	**0.8986**	**0.5114**	**0.8326**	**0.0140**	0.0066	**0.5135**	**0.7356**	**0.4363**	**0.5865**

Table 3: Results on new node attribute prediction; 'mean' indicates the matching similarity; underlined fonts indicate the best results in independent prediction; bold fonts indicate the best results among both independent and reuse predictions.

Methods	NBA				Reddit				AMiner			
	mean	median	min	max	mean	median	min	max	mean	median	min	max
Random	0.8143	<u>0.8176</u>	0.2055	**<u>1.0000</u>**	0.8317	0.8772	-0.2507	<u>0.9947</u>	0.9974	0.9983	0.9565	**<u>0.9998</u>**
FNN	0.7447	0.7877	0.2482	0.9191	0.7380	0.7850	-0.8689	0.9763	0.9903	0.9911	0.9779	0.9993
PointNet	0.6375	0.6855	0.2519	0.8299	0.6357	0.5982	-0.8676	0.9733	0.9915	0.9924	**<u>0.9794</u>**	0.9996
PROSER	<u>0.8156</u>	0.8160	<u>0.2964</u>	**<u>1.0000</u>**	<u>0.8337</u>	<u>0.8800</u>	**<u>0.0286</u>**	<u>0.9947</u>	<u>0.9975</u>	<u>0.9984</u>	0.9553	**<u>0.9998</u>**
AGATE	**0.8283**	**0.8393**	**0.3780**	0.9837	**0.8513**	**0.8908**	-0.0257	**0.9948**	**0.9977**	**0.9985**	0.9632	**0.9998**

Table 4: Mean absolute error of graph statistics prediction. From left to right, the # of new nodes, the # of lost nodes, the # of appeared links, the # of disappeared links and the # of new links, respectively.

	NBA	Reddit	AMiner
Baseline	12/10/122/115/189	43/31/2/10/38	512/-/-/-/5764
LSTM	11/ 8/108/101/180	33/67/3/42/11	618/-/-/-/3266

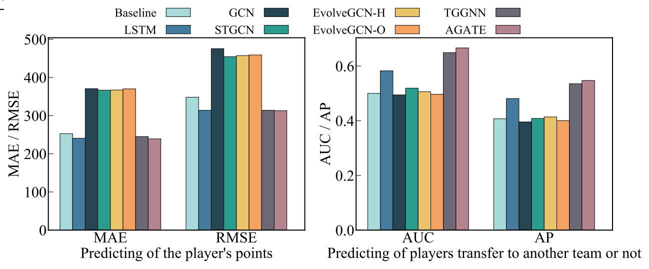


Figure 3: Existing node attribute prediction, NBA data.

Prediction on existing nodes: Table 2 presents the AUC and average precision on predictions involving existing nodes in NBA and Reddit. Note that, in AMiner, existing links and nodes do not change, hence this data is not included in this experiment. We observe that, in independent prediction, TGGNN and LSTM achieve the best performances. This result indicates that node attribute and/or topology changes affect to the future graphs in these datasets. For instance, in NBA, the scores of a player (i.e., a node attribute) may affect whether the player retires (i.e., a node loss); likewise, in Reddit, link appearance may present a periodic temporal behavior. AGATE improves on node/link prediction accuracy via reuse prediction. In particular, the accuracy of node loss prediction increases significantly compared to that of independent prediction. Interestingly, the node loss prediction is highly assisted from new node prediction. We will come back to task interdependence later, whereupon we will reconfirm that prediction accuracy improves when we can appropriately capture task interdependence.

Next, we show results for node attribute prediction. Figure 3 illustrates the MAE and RMSE on the prediction of NBA players' points (where lower is better) and AUC and Average Precision on the binary prediction of whether a player transfers to another team (where higher is better). LSTM and TGGNN perform well, as they do in node/link prediction. However, other dynamic graph GNNs cannot predict player's points and fare worse than the baseline. AGATE enhances accuracy through reuse prediction. In player's point prediction, AGATE reuses the results of LSTM and all sub-prediction tasks; in team transfer prediction, it reuses the results of TGGNN and those regarding links connected to new nodes.

Prediction on new nodes: Table 3 shows results on the prediction of new node attributes. PROSER and Random outperform FNN and PointNet. POSER and Random output node attributes appearing in the previous time step without modification, whereas FNN and PointNet modify them using neural networks. This result suggests that it is difficult to learn how the attributes of new nodes change, even while they are similar to those of new nodes at the previous time step. Further, PROSER surpasses Random, as it avoids sampling nodes that do not contribute to the matching similarity, and thus cosine similarity significantly improves overall. AGATE improves the matching similarity by means of reuse prediction based on supervised learning on the correspondences between predicted and real new node attributes obtained from the independent prediction. In reuse prediction, AGATE trains on the new nodes as existing nodes, hence it manages to improve on matching similarity.

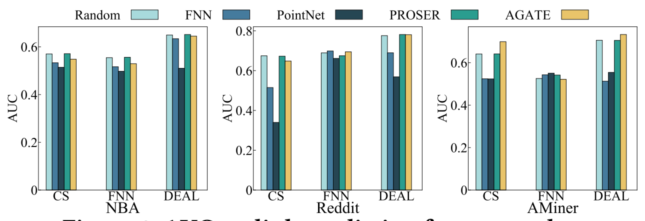


Figure 4: AUC on link prediction for new nodes.

Figure 4 shows results on the prediction of link connected to new nodes. We predict links by the methods on the horizontal axis, each following upon the results on new node attribute prediction using the methods indicated in the bars. The results are generally good when using the previous results of Random, PROSER, and AGATE, which perform well in new node attribute prediction. Among link prediction methods themselves, DEAL outperforms others.

6.4 Task interdependence analysis

Here, we examine how AGATE improves the accuracy of sub-prediction tasks by reusing the results of other tasks. Figures 5 and 6 show the difference between the AUC of independent and reuse predictions on node loss and link appearance, respectively. Task names on the horizontal axis indicate the reused task; 'all' denotes the reuse of all tasks. Each method and data reveals a different task interdependence. For example, the accuracy of STGCN improves when it uses results from all sub-prediction tasks with NBA data, but only slightly so with Reddit data. The accuracy of TGGNN improves when it uses results from new node prediction. Since TGGNN already achieves high accuracy in independent prediction, this gain is small. Still, these results confirm our hypothesis that leveraging task interdependence improves performance. In particular, the tasks of new node attribute/link prediction, though not studied previously, exercise a big impact upon other tasks.

Figures 10–12 show the difference between average prediction of independent and reuse predictions on node loss, link appearance, and link disappearance predictions, respectively.

These results are similar tendencies to Figures 5–**??**, respectively. The results of new node prediction has also large impact in average precision as same as AUC. The difference between the results of average precision and AUC is that the average precision of LSTM in the link appearance prediction on Reddit significantly decreases. This indicates that the reuse prediction in LSTM for the link appearance prediction does not work well on Reddit. While, we can see that the reuse prediction often performs well in sub-prediction tasks in all datasets.

We also examine the impact of reuse iterations, varying the number of repetitions i in Figure 2. Figure 7 shows the accuracy of sub-prediction tasks vs. i, where 'ind' indicates the accuracy of independent prediction; the reuse prediction improves prediction accuracy if that increases compared to the accuracy of 'ind'. We reuse the results of all sub-prediction tasks. Accuracy increases perceptibly in most tasks from 'ind' to 'reuse1'. In node loss prediction, it keeps increasing with reuses; in other tasks, accuracy stays stable or even falls after 'reuse1', due to the accumulation of errors.

6.5 Scalability

Figure 8 shows inference times on real graphs. All methods take less than 10 seconds; TGGNN needs slightly larger time than others, while PROSER efficiently predicts attributes of new nodes. Figure 9 shows the inference time on synthetic graphs generated by the BA model, varying with the number of nodes. It is worth noting that the number of nodes in the experimental studies of prior works is less than 100,000. The inference time grows linearly with the number of nodes. All methods we use for lost and new node prediction are scalable to large graphs. Contrariwise, predicting link appearances and disappearances is inherently hard to scale to large graphs, for all methods, as it requires prediction on all vertex pairs.

7 RELATED WORK

We now review previous work on graph prediction. Table 5 outlines a summary of existing methods, categorized in four groups: (1) *static embedding* methods [9, 33], (2) *dynamic embedding* methods [7, 8], (3) *static graph neural network* methods [10, 11, 13, 18, 34, 41], and (4) *dynamic graph neural network* methods [17, 19, 23, 28, 36, 37, 39]. Embedding methods first derive an embedding independently of target tasks, and then build a model for each target task using the embedding as input; on the other hand, graph neural network methods directly build a model for each target task. Each method makes its own assumptions about graph properties and inference; such methods apply to some, but not all prediction tasks. Besides, to our knowledge, previous work do not exploit task interdependence; they address each sub-prediction task individually.

There are two types of graph prediction tasks: static and dynamic. Most methods assume that the underlying graph is static, and predict or reconstruct elements, such as links and attributes [9, 24]. Yet most real-world systems and data, such as social network services and citation networks, are dynamic. While it may be possible to apply static graph methods [20] to dynamic graphs while ignoring the temporal evolution, this approach is sub-optimal [38]. Learning on dynamic graphs has been recently studied, yet such studies are usually limited to the *transductive* case involving *observed* elements [7, 8, 17, 37, 39]. Unfortunately, such approaches are insufficient for real-world settings in which a graph evolves with links and new nodes appearing at any time. Some methods [23, 28] support the inductive setting, involving nodes *unobserved* in training. Besides, some methods assume *continuous-time* dynamic graphs that are continuously modified [2, 15, 22, 27, 32, 38]; however, methods for continuous-time dynamic graphs do not support prediction for discrete-time dynamic graphs, and vice versa.

Among existing methods, only EvolveGCN [23] predicts changes of nodes, links, and node attributes, hence supports time-evolving attributed graphs, notwithstanding new node appearances. TGGNN supports what EvolveGCN supports, yet differs in two ways. First, EvolveGCN uses graph convolutional networks (GCNs) [13]; upon a significant change in graph structure, GCNs can neither recognize that a node may have a different aggregation vector, nor distinguish nodes having the same vector. EvolveGCN recursively updates GCN weights using a recurrent neural network (e.g., GRU and LSTM), yet retains the GCN shortcomings. Contrariwise, TGGNN learns node embeddings by aggregating hidden node vectors within k hops at each time step, while computing aggregation weights for hidden

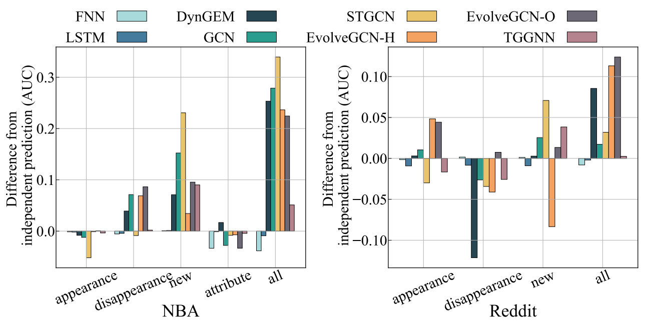


Figure 5: Reuse gain on node loss prediction.

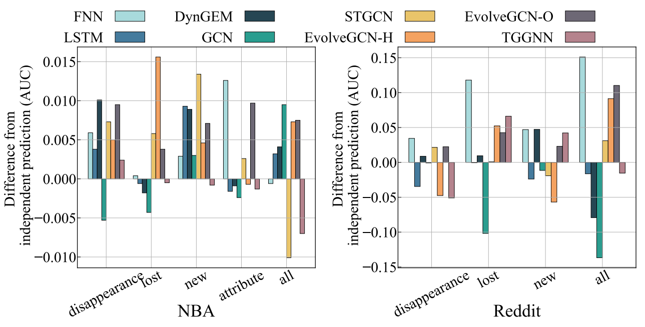
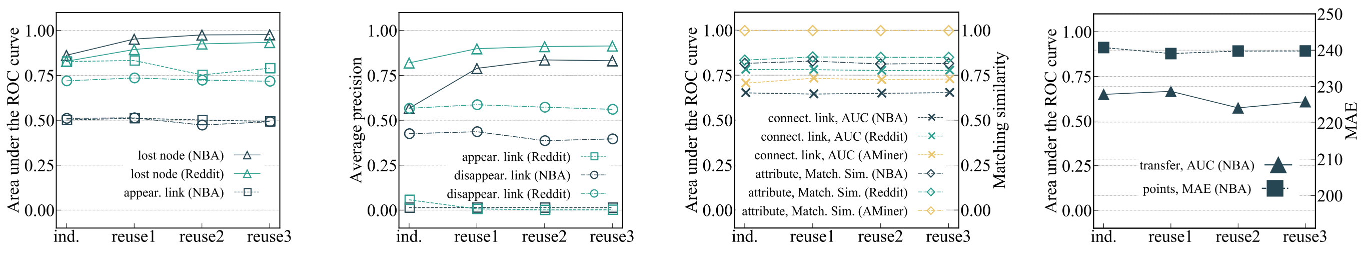


Figure 6: Reuse gain on link appearance prediction.

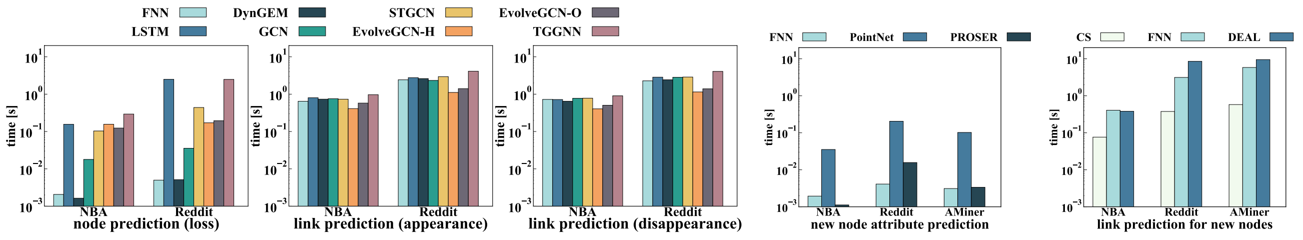


(a) node/link prediction, exist. nodes (legend split among two figures)

(b) attribute/link prediction, new nodes

(c) attribute prediction, exist. nodes

Figure 7: Impact of reuse iterations.



(a) Existing node prediction (node loss, link appearance, and link disappearance)

(b) New node (attributes and links of new nodes)

Figure 8: Inference time on real graphs.

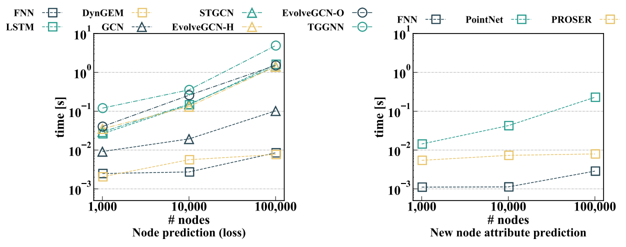


Figure 9: Inference time on synthetic graphs.

vectors. Second, EvolveGCN incorporates node attributes in hidden vectors at each time step, whereas TGGNN captures temporal changes in node attributes via graph annotations handled separately from graph structure, enhancing attribute prediction accuracy. In effect, TGGNN handles graph structure and node attribute changes more effectively than EvolveGCN.

EvoNet [35] and open-world knowledge-graph (OWKG) completion [29] tackle problems different from ours. EvoNet generates a future graph without correspondences between nodes at different time steps. OWKG completion predicts unseen nodes from the text data of existing ones. Contrariwise, our predictions track evolving nodes without specifically relying on additional text data; to our knowledge, no existing method supports such a task.

8 CONCLUSION

We introduced the problem of *holistic* prediction on a time-evolving attributed graph, including new node appearances, and proposed AGATE, a framework that exploits task interdependence. To build AGATE, we developed two novel methods, TGGNN for predictions on existing nodes, and PROSER, for predictions about new nodes and their attributes. Our experimental study showed that prediction accuracy largely improves by exploiting task interdependence. Future work may employ auto-ML techniques to discern task interdependence that may enhance performance.

REFERENCES

[1] Aminer dataset. [n.d.]. https://www.aminer.cn/citation.

[2] Nikolaos et al Bastas. 2019. evolve2vec: Learning Network Representations Using Temporal Unfolding. *The MultiMedia Modeling* (2019), 447–458.

[3] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv:1406.1078* (2014).

[5] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *arXiv:1612.08083* (2016).

[6] Github repository of AGATE. [n.d.]. https://github.com/hsack6/AGATE.

Table 5: A summary of existing works and their characteristics.

	methods	graph property			inference		prediction target				
		attribute	temporal	notation	transductive	inductive	attribute	appeared link	disappeared link	lost node	new node
embedding	node2vec [9]	✘	static	$G = (V, E)$	✔	✘	✘	✔	✔	✔	✘
	VERSE [33]	✘	static	$G = (V, E)$	✔	✘	✘	✔	✔	✔	✘
	G2G [3]	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	GraphSAINT [40]	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	DynGEM [8]	✘	dynamic	$G_t = (V_t, E_t)$	✔	✘	✘	✔	✔	✔	✘
	dyngraph2vec [7]	✘	dynamic	$G_t = (V_t, E_t)$	✔	✘	✘	✔	✔	✔	✘
graph neural network	GCN [13]	✔	static	$G = (V, E, X)$	✔	✘	✘	✔	✔	✔	✘
	GGNN [18]	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	GAT [34]	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	GraphSAGE [10]	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✔	✔	✘
	SEAL [41]	✔	static	$G = (V, E, X)$	✔	✘	✘	✔	✘	✘	✘
	DEAL [11]	✔	static	$G = (V, E, X)$	✔	✔	✘	✔	✘	✘	✘
	SAPE [17]	✔	dynamic	$G_t = (V, E_t, X_t)$	✔	✘	✘	✘	✔	✘	✘
	STAR [36]	✔	dynamic	$G_t = (V, E_t, X_t)$	✔	✘	✔	✔	✔	✔	✘
	DCRNN [19]	✔	dynamic	$G_t = (V, E, X_t)$	✔	✘	✔	✔	✔	✔	✘
	STGCN [39]	✔	dynamic	$G_t = (V, E, X_t)$	✔	✘	✔	✔	✔	✔	✘
	TRRN [37]	✔	dynamic	$G_t = (V, E_t, X_t)$	✔	✘	✔	✔	✔	✔	✘
	DySAT [28]	✘	dynamic	$G_t = (V_t, E_t)$	✔	✔	✘	✔	✔	✔	✘
	DANE [16]	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✘	✔	✘	✘	✔	✘
	EvolveGCN [23]	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✔	✔	✔	✔	✔	✘
ours	TGGNN	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✔	✔	✔	✔	✔	✘
	PROSER	✔	dynamic	$G_t = (V_t, X_t)$	✔	✔	✔	✘	✘	✘	✔
	AGATE	✔	dynamic	$G_t = (V_t, E_t, X_t)$	✔	✔	✔	✔	✔	✔	✔

[7] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187, 104816 (2020).

[8] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv:1805.11273* (2018).

[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.

[10] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.

[11] Yu Hao, Xin Cao, Yixiang Fang, Xike Xie, and Sibo Wang. 2020. Inductive Link Prediction for Nodes Having Only Attribute Information. In *IJCAI*. 1209–1215.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[14] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. 2018. Community interaction and conflict on the web. In *WWW*. 933–943.

[15] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *KDD*. 1269–1278.

[16] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *CIKM*. 387–396.

[17] Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. 2019. Predicting Path Failure In Time-Evolving Graphs. In *KDD*. 1279–1289.

[18] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *ICLR*.

[19] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.

[20] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *The JASIST* 58, 7 (2007), 1019–1031.

[21] NBA dataset. [n.d.]. https://www.basketball-reference.com.

[22] G. H. Nguyen, J. Boaz Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. 2018. Dynamic Network Embeddings: From Random Walks to Temporal Random Walks. *The IEEE International Conference on Big Data* (2018), 1085–1092.

[23] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*. 5363–5370.

[24] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD*. 701–710.

[25] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*. 77–85.

[26] Reddit dataset. [n.d.]. http://snap.stanford.edu/data/soc-RedditHyperlinks.html.

[27] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv:2006.10637* (2020).

[28] Aravind Sankar, Y. Wu, L. Gou, Wei Zhang, and H. Yang. 2020. Dynamic Graph Representation Learning via Self-Attention Networks. In *WSDM*. 519–527.

[29] Baoxu Shi and Tim Weninger. 2018. Open-world knowledge graph completion. In *AAAI*. 1957–1964.

[30] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnet-Miner: Extraction and Mining of Academic Social Networks. In *KDD*. 990–998.

[31] Steven L Tanimoto, Alon Itai, and Michael Rodeh. 1978. Some matching problems for bipartite graphs. *J. ACM* 25, 4 (1978), 517–525.

[32] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.

[33] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *WWW*. 539–548.

[34] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[35] Changmin Wu, Giannis Nikolentzos, and Michalis Vazirgiannis. 2020. EvoNet: A neural network for predicting the evolution of dynamic graphs. In *International Conference on Artificial Neural Networks*. 594–606.

[36] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. 2019. Spatio-Temporal Attentive RNN for Node Classification in Temporal Attributed Graphs. In *IJCAI*. 3947–3953.

[37] Dongkuan Xu, Junjie Liang, Wei Cheng, Hua Wei, Haifeng Chen, and Xiang Zhang. 2021. Transformer-Style Relational Reasoning with Dynamic Memory Updating for Temporal Network Modeling. In *AAAI*, Vol. 35. 4546–4554.

[38] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *arXiv:2002.07962* (2020).

[39] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*. 3634–3640.

[40] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[41] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NIPS*. 5165–5175.

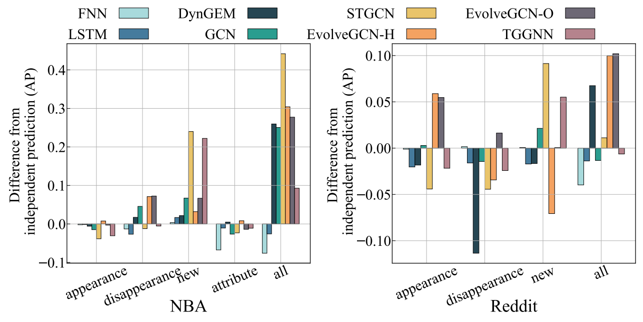


Figure 10: Reuse gain on node loss prediction (Average precision)

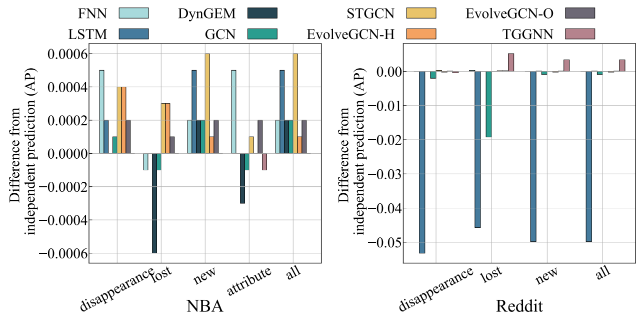


Figure 11: Reuse gain on link appearance prediction (Average precision)

A HYPER PARAMETERS

We describe hyper parameter tuning in each method. Please see our source codes in detail. In all experiments, we run the model with 100−1000 training iterations, 10−100 early stopping patience, 1 or 2 batch sizes, and a learning rate of 0.01 with Adam as the optimizer. For methods on temporal graphs, we use 3, 3, and 5 as L for NBA, Reddit, and Aminer, respectively.

For all models, the hidden state's size is the same as the node attribute dimension, only for LSTM is three times the dimension of node attributes. The output layer is a fully-connected layer for all models, and its activation function is a softmax for multi-class classification, a sigmoid for binary classification, and none for regression.

The architecture and hyper parameters in DynGEM[1], DEAL[2], and EvolveGCN[3] are the same to the author's implementation. The embedding size of DynGEM is the same as the node attribute dimension. Hyper parameters in PointNet generally follow the setting on original PointNet but we modify the number of units depending on input and output sizes. FNN, LSTM, and GCN have a single layer.

In TGGNN, its architectural hyper parameters have been optimized on the Reddit dataset and are then reused for NBA. The time-directed convolution layers of TGGNN consists of filters whose

[1]https://github.com/palash1992/DynamicGEM
[2]https://github.com/working-yuhao/DEAL
[3]https://github.com/IBM/EvolveGCN

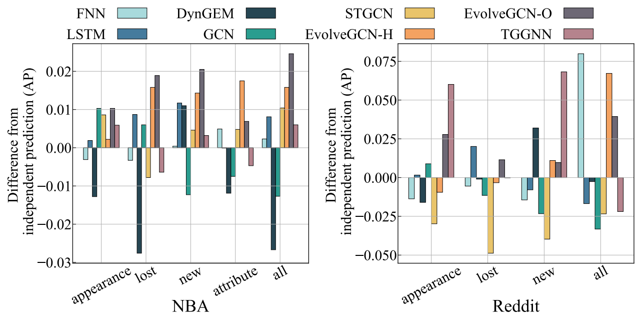


Figure 12: Reuse gain on link disappearance prediction (Average precision)

kernel size $K = 2$ (same as STGCN). The number of aggregate updates by the propagation model is five, which is the same number of times as the pytorch implementation of GGNN for bAbI task 15[4]. ST-Gate is repeated one time. The hidden state's size $\hat{d}$ is the same as attribute dimension.

B SELECTED MODELS OF AGATE

We summarize the models that we use for evaluating AGATE. Tables 6–8 shows models that AGATE uses in NBA, Reddit, and AMiner. We select models that achieve the best performance of validation in each sub-prediction task. Each number is corresponding to the task number in Figure 2. For the task numbers 1–5, we use LSTM in all datasets. The missing numbers (e.g., 11 in Table 7) are regardless tasks to the datasets (e.g., attribute prediction in Reddit).

[4]https://github.com/chingyaoc/ggnn.pytorch

Table 6: Methods used in AGATE of NBA Dataset

Independent prediction										
Task number	6	7	8	9	10	11				
						team (transfer or not)	team (which team)	position	points	age
Method	PROSER	DEAL	TGGNN	GCN	DynGEM	TGGNN	GCN	Baseline	LSTM	TGGNN

Repeat prediction										
Task number	12	13		14	15					
		existing	new		team (transfer or not)	team (which team)	position	points	age	new
Method	TGGNN	FNN	DEAL	DynGEM	TGGNN	GCN	Baseline	LSTM	Baseline	LSTM

Table 7: Methods used in AGATE of Reddit Dataset

	Independent prediction					Repeat prediction				
Task number	6	7	8	9	10	12	13		14	15
							existing	new		
Method	PROSER	DEAL	TGGNN	LSTM	LSTM	STGCN	TGGNN	DEAL	LSTM	FNN

Table 8: Methods used in AGATE of AMiner Dataset

	Independent prediction		Repeat prediction	
Task number	6	7	13	15
Method	PROSER	DEAL	DEAL	FNN